Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Nejc Smolej

Sistem za preverjanje prisotnosti na preverjanju znanja

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: doc. dr. Dejan Lavbič

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Preverjanja znanja, ki se jih udeležuje večje število študentov, so lahko z vidika nadzornikov pogosto časovno zelo zahtevna. Pred začetkom preverjanja znanja je potrebno namreč preveriti identiteto vsakega študenta, da ne pride do poskusov zlorabe. Ta del procesa je možno avtomatizirati in izboljšati, zato v okviru diplomske naloge predlagajte in implementirajte prototip takšnega sistema, ki bo informacijsko podprl začetni del preverjanja znanja, t.j. identifikacijo študenta, določitev sedežnega reda in obvladovanje vseh prijavljenih na preverjanje znanja. Predlagani prototip naj temelji na najnovejših tehnologijah in ga ovrednotite na realnem primeru preverjanja znanja ter ga primerjajte z ročnim pristopom.

Zahvalil bi se družini in prijateljem, ker me podpirajo pri delu. Prav tako pa tudi mentorju, ker mi je nudil podporo, znanje in odlične ideje za boljši razvoj sistema.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
NFC	near field communication	komunikacija kratkega dosega
RFID	radio frequency identification	radiofrekvenčno prepoznavanje
BLE	Bluetooth low energy	nizko potrošni Bluetooth
REST	representational state transfer	reprezentacijsko izmenjavanje stanja
XML	extensible markup language	razširljivi označevalni jezik
JSX	JavaScript syntax extension	razširjena sintaksa Javascripta
API	application programming interface	programski vmesnik
HTTP	hypertext transfer protocol	protokol za prenos hiperteksta
NoSQL	non structured query language	nestrukturiran povpraševalni jezik
JSON	JavaScript object notation	zapis Javascript objekta
URL	uniform resource locator	enotni naslov vira
TLS	transport layer security	varnost transportnega sloja
JWT	JSON web token	JSON spletni žeton
HMAC	hash-based message authentication code	koda za preverjanje pristnosti sporočila na podlagi zgoščevalne funkcije
SWOT	strengths, weaknesses, opportunities, and threats analysis	analiza prednosti, pomanjkljivosti, priložnosti in groženj

Povzetek

Naslov: Sistem za preverjanje prisotnosti na preverjanju znanja

Avtor: Nejc Smolej

Diplomska naloga rešuje problem, kako učinkoviteje preverjati avtentikacijo posameznih študentov. S pomočjo mobilne aplikacije, ki si jo bo vsak študent namestil na svoj osebni telefon in nadzorno ploščo, ki jo bo profesor imel naloženo na svojem računalniku bo preverjanje prisotnosti enostavno. Sprva bo vsak študent dobil dodeljen sedež, na katerega se bo moral usesti. Ko bo prišel na svoj sedež, bo s slikanjem svojega obraza in s podpisom na mobilno napravo postopek končan. Medtem pa bo imel profesor s pomočjo nadzorne plošče pregled nad aktivnostmi študentov. Videl bo sliko študenta, ki jo bo lahko primerjal z osebno izkaznico. Prav tako bo videl podpis, katerega bo lahko primerjal s prejšnjimi podpisi. Tako bo avtentikacija izboljšana in študentje ne bodo mogli goljufati na preverjanju znanja, medtem ko bo profesorju olajšano delo preverjanja prisotnosti na preverjanju znanja.

Ključne besede: nadzorna plošča, mobilna aplikacija, BLE Beacon, preverjanje prisotnosti.

Abstract

Title: System for presence verification of students

Author: Nejc Smolej

My thesis solves problem how to effectively verify students in a class. With students having installed mobile application and professor having a dashboard, this would be possible. Each time the exam is about to start the students will receive the seat they have to take. Once the student get to the seat they will have to take their picture and sign themselves. Finally they will get a code with which the student can start an exam. All of this actions will be monitored and displayed on the dashboard, which will be used by professor. I think that with the usage of mobile application the authentication of the students would be improved.

Keywords: dashboard, mobile application, BLE Beacon, presence verification.

Poglavje 1

Uvod

1.1 Motivacija

V času izpitov se na številnih fakultetah profesorji in asistenti soočajo z istim problemom preverjanja prisotnosti študentov na preverjanjih znanja. Ko profesor vstopi v predavalnico, mora najprej preveriti, ali študentje sedijo na pravih mestih. Zatem mora preveriti, ali so študentje prisotni in kasneje še preveriti njihove študentske izkaznice. Za profesorja so to zgolj časovno potratne naloge. Ne smemo pa pozabiti na študente, ki so nemalokrat žrtev presedanja zaradi nejasnosti sedežnega reda.

Živimo v dobi mobilnih naprav. Po zadnjih podatkih naj bi bilo v Sloveniji 2.4 miljone mobilnih uporabnikov [1]. Kar pomeni, da si v povprečju vsak Slovenec lasti več kot eno mobilno napravo. Prav tako pa število mobilnih aplikacij vsakodnevno močno narašča. Več kot očitno je, da so mobilne naprave prihodnost in last vsakega študenta.

Zato je bil v okviru diplomskega dela razvit sistem, ki bo s pomočjo mobilne aplikacije študenta, BLE Beacona in nadzorne plošče profesorja povsem avtomatiziral in izboljšal preverjanje prisotnosti na preverjanjih znanja. Mobilna aplikacija študentu omogoča potrjevanje prisotnosti, pregled prihajajočih preverjanj znanj in pregled ocen. Pregledna plošča pa bo profesorju v času preverjanja znanja omogočala pregled nad prisotnostjo študentov.

1.2 Sorodna dela

V zadnjih letih je opazen interes posameznikov za izboljšavo sistema preverjanja prisotnosti na preverjanjih znanja. Eden izmed njih je Dean Koštomaj, ki je za avtentikacijo študentov uporabil izkaznico NFC (ang. Near-field communication) [2]. Diplomska naloga študenta na Fakulteti za računalništvo in informatiko opisuje rešitev oziroma aplikacijo, ki bi si jo namestil profesor. Profesor bi skeniral študentske izkaznice študentov, ki imajo vgrajen NFC-sistem. Na podoben način se je istega problema lotil Dejan Lebar z diplomskim delom Prisotnost pri pouku s pomočjo RFID [3]. Za potrebe delovanja sistema je izdelal modul RFID (ang. Radio-frequency identification) s čitalcem čipov. Tako bi se študentje potrdili s pomočjo čipa, ki bi ga prislonili na čitalec na učiteljevi mizi. Pri obeh primerih smo zaznali pomanjkljivost v avtentikaciji študenta, saj si študent lahko sposodi izkaznico kolega. Nadzornik nima nadzora nad sedežnim redom, kot nima shranjenih zajetih slik študentov in izkaznic za ponovno preverjanje ujemanja. Poleg tega si povprečni študent ne lasti izkaznice s čipoma RFID ali NFC.

Uporaba BLE Beacona se je močno povečala v razvoju različnih sistemov za izboljšavo delovanja tega. V svetu trgovcev se uporablja za komunikacijo s kupci, v nekaterih trgovinah in stavbah za pomoč pri pozicioniranju znotraj stavbe. Eno izmed takih podjetij je IndoorAtlas [4]. S pomočjo magnetnega polja in BLE Beaconov lahko pozicionirajo osebo znotraj stavbe tudi do 1 metra natančno. Gre za eno izmed najbolj ambicioznih podjetij, saj je ta podatek natančnosti najboljši glede na zahteve za delovanje. Vse, kar mobilna naprava potrebuje sta sistem Bluetooth in kompas oziroma giroskop. V prihodnosti bi nam pri razvijanju našega sistema lahko prav prišla njihova tehnologija.

1.3 Cilj diplomske naloge

Cilj diplomske naloge je preveriti, ali bi se sistem lahko obnesel pri pravem preverjanju znanja. Glavni del sistema sestavlja mobilna aplikacija, ki je

glavni prototip. Delovanje mobilne aplikacije in celotnega sistema na preverjanju znanja smo lahko preverili na testni množici 52 študentov pri predmetu Osnove informacijskih sistemov. Mobilna aplikacija mora biti narejena kar se da preprosto za uporabo, saj je glavni cilj poenostavljanje opravil in s tem pohitriti postopek preverjanja, medtem ko je treba profesorju omogočiti učinkovit pregled nad stanji študentov v času preverjanja znanja.

Za razvoj sistema so bile uporaljene tehnologije, ki jih uporabljajo večje tehnološke kooperacije po svetu. V nadaljevanju bo v poglavju *Pregled uporabljenih tehnologij* prikazana shema sistema, kasneje pa bo podrobneje predstavljena uporabljena tehnologija za vsako komponento sistema posebej.

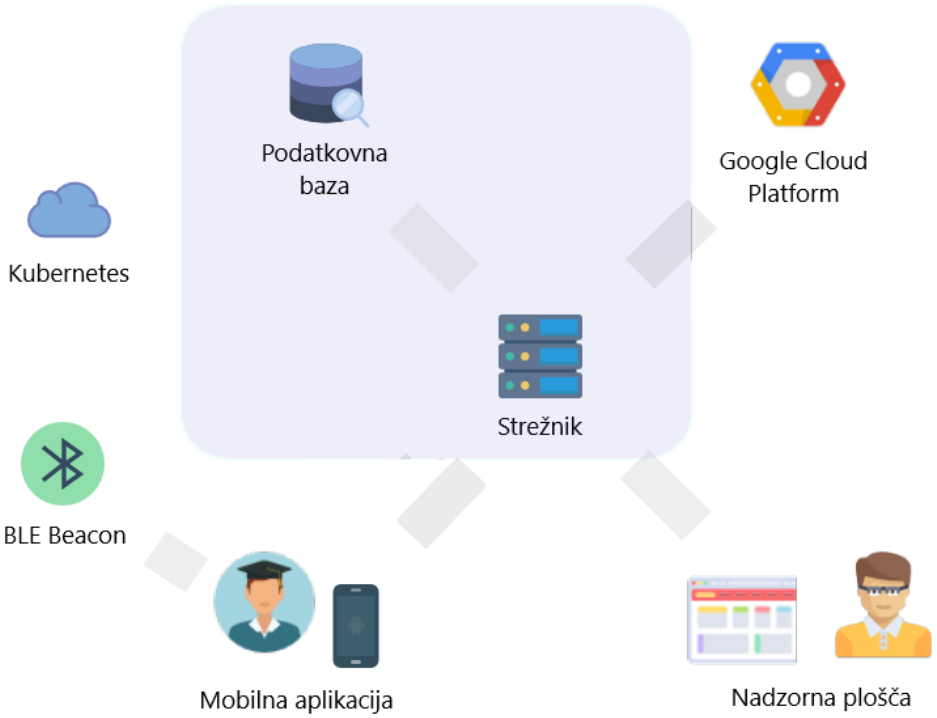
Tretje poglavje *Implementacija sistema* vsebuje postopek delovanja sistema za uporabnika in postopek razvoja za nadzornika. Skupaj s postavitvijo strežnika v oblak in shemo podatkovne baze.

Celoten postopek in priprava na testiranje sta opisana v četrtem poglavju *Testiranje*.

Poglavje 2

Pregled uporabljenih tehnologij

2.1 Shema rešitve



Slika 2.1: Shema rešitve

Celotna rešitev, prikazana na Sliki 2.1 je sestavljena iz 7 komponent:

- Kubernetes,
- podatkovna baza,
- Google Cloud Platform,
- strežnik,
- BLE Beacon,
- mobilna aplikacija,
- nadzorna plošča.

Osrednja komponenta je strežnik, ki povezuje mobilno aplikacijo in nadzorno ploščo s podatkovno bazo. Kubernetes skrbi za skalabilnost in delovanje podatkovne baze in strežnika, medtem ko Google Cloud Storage služi kot hramba slikovnih datotek. Nadzorno ploščo bo pregledoval profesor in s tem preverjal stanje avtentikacije študentov. Vsak študent bo imel mobilno aplikacijo, ki pa se bo morala v času preverjanja znanja povezati z BLE Beaconom in s tem potrditi, da se študent nahaja v predavalnici.

2.2 Mobilna aplikacija

2.2.1 React Native

Za glavno tehnologijo pri razvoju aplikacije je bila uporabljena knjižnica React Native. React Native je Javascript ogrodje (ang. framework) za pisanje pravih (ang. natively) upodobljenih mobilnih alikacij za platformi Android in iOS. Zgrajena je na osnovi Reacta, Facebookove Javascript knjižnice za gradnjo uporabniških vmesnikov za spletne strani brskalnikov [5]. Tako je React Native za razliko od Reacta namenjen gradnji mobilnih aplikacij. Gre za knjižnico, ki je vse bolj uporabljena v podjetjih, kot so Facebook, Airbnb, Instagram in SoundCloud.

Podobno kot React za spletne strani, so aplikacije z React Nativom napisane kot mešanica Javascripta in XML-a, znana tudi kot JSX. V ozadju se skriva most (ang. bridge) React Native, ki sproži pravo (ang. native) prevajanje (ang. rendering) API v Objective-C za iOS ali v Javo za Android.

Slika 2.2 prikazuje kodo, ki v mobilni aplikaciji poganja pogled z vstopno kodo za študenta za začetek preverjanja znanja.

```
class CodeView extends Component {
  constructor() {
    super();
    this.state= {
      opacity: new Animated.Value(0),
      textOpacity: new Animated.Value(0)
    }
  }
  componentDidMount() {
    this.props.getCode();
    Animated.timing(this.state.opacity, {toValue: 1, duration: 2500}).start();
    Animated.timing(this.state.textOpacity,{ toValue: 0.9, duration: 800}).start();
  }

  render() {
    return (
      <View style={{flex: 1}}>
        <View style={styles.view}>
          <Animated.Text style={[{opacity: this.state.textOpacity}, styles.fontR]}>
            Vaša koda je
          </Animated.Text>
          <Animated.View style={[styles.codeView,{opacity: this.state.opacity}]}>
            <Text style={styles.code}>
              { this.props.currentCode }
            </Text>
          </Animated.View>
          <Text style={[{fontSize: 10},styles.fontR]}>
            Srečno na izpitu! :)
          </Text>
        </View>
        <TouchableOpacity style={styles.finish} onPress={() => global.nav.resetTo({key: 'MAIN'})}>
          <Text style={[{fontSize: 18},styles.fontR]}>
            KONČAJ
          </Text>
        </TouchableOpacity>
      </View>
    );
  }
}
```

Slika 2.2: Primer kode z uporabo knjižnice React Native

2.3 Nadzorna plošča

2.3.1 React

React je Facebookova Javascript knjižnica za gradnjo uporabniških vmesnikov za spletne brskalnike [7]. Ena glavnih prednosti Reacta je možnost gradnje različnih komponent, ki skupaj sestavljajo kompleksne uporabniške vmesnike. Vsaka komponenta lahko sprejema podatke, dostopne na *this.props*, prav tako pa ima svoje stanje, dostopno v spremenljivki *this.state*.

2.3.2 Electron

Electron je nova odprtokodna (ang. open-source) knjižnica, narejena s strani podjetja Github, ki razvijalcem omogoča gradnjo namiznih aplikacij z jezikom Javascript [8].

2.4 BLE Beacon

Slika 2.3: iBKS Plus

BLE Beacon ni nič več kot zelo majhen računalnik, ki oddaja signale Bluetooth. Te signale lahko zaznajo vse naprave, tako mobilne kot različni računalniki z vgrajenim sistemom Bluetooth. Signal Beacona lahko sega tudi do 100m daleč, prav tako pa lahko preračuna, koliko je naprava oddaljena od vira Bluetootha [9]. Kar pomeni, da lahko sprožimo različne dogodke glede na moč signala oziroma oddaljenost naprave od BLE Beacona.

Na Sliki 2.3 je prikazan BLE Beacon iBKS Plus, uporabljen za testiranje, ki ga je na trgu možno dobiti za zgolj 20 €, njegova povprečna življenska doba pa je kar 70 mesecev.

2.5 RethinkDB

RethinkDB je odprtokodna podatkovna baza NoSQL. Podatkovna baza hrani dokumente JSON ter ostalim spletnim aplikacijam omogoča poizvedovanje v realnem času. RethinkDB prav tako omogoča porazdelitev podatkov. S tem veliko pridobi na skalabilnosti ter hitrosti operiranja s podatki [10].

Odmika se od tipične podatkovne baze z implementacijo novega dostopnega modela. Namesto, da spletna aplikacija poizveduje po spremembah v bazi, lahko razvijalec zahteva od baze, da v trenutku spremembe v bazi ali tabeli to nemudoma sporoči aplikaciji. Ena največjih konkurenc pri podatkovnih bazah JSON NoSQL je veliko bolj prepoznavna podatkovna baza MonogoDB. A prav zaradi zadnje omenjene prednosti smo se odločili za omenjeno bazo RethinkDB.

Baza RethinkDB ima tudi svoj poizvedovalni jezik ReQL, ki omogoča obvladovanje in nadziranje baze kot tudi poizvedovanje podatkov. Kot primer je priložena na Sliki 2.4 preprosta poizvedba, ki predstavlja največjo prednost pred ostalimi podatkovnimi bazami poizvedovanje v realnem času.

```
r.db('ois').table('students').changes()
```

Slika 2.4: Poizvedba v realnem času

Ko se zgodi sprememba v tabeli *students*, spletna aplikacija prejme dokument JSON prikazan na Sliki 2.5 s priloženo prejšnjo vrednostjo *old val* in novo vrednostjo *new val*. V tem primeru je baza zaznala spremembo v objektu, ko je študent posodobil URL (ang. Uniform Resource Locator) slike osebne izkaznice.

```
{
  "new_val": {
    "id": "63130222",
    "firstName": "Nejc",
    "lastName": "Smolej",
    "idCard": "https://i.pinimg.com/originals/45.jpg"
  },
  "old_val": {
    "id": "63130222",
    "firstName": "Nejc",
    "lastName": "Smolej",
    "idCard": ""
  }
}
```


Slika 2.5: Odgovor na poizvedbo navedeno v Sliki 2.4

2.6 Strežnik

2.6.1 Node.js

Node.js je odprtokodno pogonsko okolje za razvijanje strežnikov in spletnih aplikacij, napisanih z jezikom Javascript. Nudi veliko zbirko knjižnic in modulov, ki vsakemu razvijalcu močno poenostavi gradnjo spletnih aplikacij [11].

Za postavitev strežnika smo uporabili ogrodje Express [12], ki močno olajša delo razvijalcu pri implementaciji strežnika Node.js. Kreiranje strežnika je preprosto in ga je moč kreirati zgolj s tremi osnovnimi ukazi:

1. *const server = express();*
2. *server.get(route, callback)*
3. *server.listen(port, callback)*

Sprva naredimo instanco strežnika in ga shranimo v spremenljivko *server*. Strežniku nato določimo pot (ang. route) in vrata (ang. port), na katerih bo dostopen. Za potrebe delovanja je koda za kreiranje strežnika primernega za sistem videti tako, kot na Sliki 2.6.

```
const server = express();

export const connect = async () => {...
};

connect();

passport.use(jwtStrategy);

server.use(passport.initialize());

morgan.token('graphql-query', morganGraphql);

server.use(morgan(':method :url :status :response-time[0]ms :date[iso] :graphql-query'));

server.use(bodyParser.json());

server.use(cors());

var storage =   multer.diskStorage({...
});

const upload = multer({ storage });

server.use(upload.single('file'));

server.use('/auth', graphqlHTTP({...
}));

server.get('/', (req: any, res: any) => {...
});

server.post('/test', (req: any, res: any) => {...
});

server.use('/graphql', passport.authenticate('jwt', { session: false }), graphqlHTTP({...
}));

server.all('*', (req: any, res: any) => {...
});

server.listen('3000', () => {
  console.info('Server is listening on port 3000!');
});
```

Slika 2.6: Kreiranje strežnika Node.js

2.6.2 GraphQL

Facebook je leta 2012 razvil svoj poizvedovalni jezik, imenovan GraphQL. Leta 2015 ga je odprl za javnost in od takrat naprej ga uporabljajo vsa večja podjetja, kot so Pinterest, Twitter, Github, Shopify, Sky itd. [13]. Nudi alternativo REST in podobnim storitvam.

Odjemalcu omogoča definiranje strukture podatkov, ki jih potrebuje, strežnik

pa odgovori s podatki in strukturo, ki so bili zahtevani. GraphQL močno razbremeni delo strežnika in pohitri njegovo delovanje [14]. Tako lahko odjemalec sam določa, katere podatke bi rad prejel, ne da bi karkoli spreminjal na strani strežnika. Kot primer sta priložena poizvedba in odgovor strežnika na Sliki 2.7.

```
mutation {
  login(email: "ns1024@student.uni-lj.si", password: "test123") {
    user {
      id
      firstName
      email
      exam_results {
        result
        lost_focus
      }
    }
  }
}
```

```
"login": {
  "user": {
    "id": "63130222",
    "firstName": "Nejc",
    "email": "ns1024@student.uni-lj.si",
    "exam_results": [
      {
        "result": "52,23",
        "lost_focus": null
      },
      {
        "result": "NA",
        "lost_focus": null
      },
      {
        "result": "50,91",
        "lost_focus": "4"
      },
      {
        "result": "62,23",
        "lost_focus": null
      }
    ]
  }
}
```

Slika 2.7: Zahteva odjemalca (levo) in odgovor strežnika (desno)

2.7 Kubernetes

Kubernetes je sistem, s katerim lahko avtomatiziramo uvedbo, širitev in upravljanje programskih vsebnikov, ki vse bolj prodirajo na področje virtualizacije aplikacij [15]. Omogoča, da več vsebnikov združujemo v funkcionalne celote in jih tako tudi obravnavamo pri upravljanju in selitvah med strežniki. Ker znajo gradniki med sabo komunicirati prek vmesnika API, je omogočen zelo učinkovit in preprost postopek za povečanje zmogljivosti aplikacije z dodajanjem instanc vsebnikov tam, kjer je potrebna večja zmogljivost glede na izmerjeno breme.

2.8 Google Cloud Platform

Google Cloud Platform je nadzorna plošča za računalniške storitve v oblaku (ang. cloud computing). Za potrebe razvoja sistema sta bila uporabljena dva produkta:

- Google Cloud Storage za hranjenje slik,
- Google Compute Engine za hranjenje in pogon virtualnih naprav.

2.8.1 Google Cloud Storage

Google Cloud Storage je RESTful spletna storitev za hranjenje in dostopanje do podatkov znotraj Googlove infrastrukture. Nudi visoko zmogljivost in skalabilnost podatkov z vgrajeno močno zaščito. V primerjavi s hrambo Amazonovega S3 je Googlov izdatno cenejši in občutno hitrejši pri prenosu podatkov [16].

2.8.2 Google Compute Engine

Google Compute Engine omogoča zagon virtualnih naprav na zahtevo [17]. Za potrebe zagona mora uporabnik najprej kreirati svojo sliko (ang. image)

virtualne naprave. Slika vsebuje operacijski sistem in ustrezen datotečni sistem, ki je potreben za zagon naprave (Slika 2.8).

```
FROM node

# Create server directory
RUN mkdir -p /usr/src
WORKDIR /usr/src

# Install dependencies
COPY package.json /usr/src
RUN npm install

# Copy source code
COPY . /usr/src
RUN chmod u+x /usr/src/run.sh

# Export port 3000 for outside connections
EXPOSE 3000

# Starts the server
CMD "/usr/src/run.sh"
```

Slika 2.8: Dockerfile datoteka, potrebna za zagon virtualne naprave

2.9 Ostala tehnologija

2.9.1 Axios

Axios je HTTP-odjemalec, ki se uporablja za klicanje različnih zahtev na strežnik [6]. Nudi nam API za lažje operiranje s XMLHttpRequesti. Poleg tega lahko v glavo (ang. header) zahteve dodamo različne nastavitve, ki nam olajšajo potek klicanja na strežnik. Uporabil sem ga pri razvoju mobilne aplikacije kot tudi pri razvoju nadzorne plošče.

Axios je bil za potrebe pridobivanja podatkov s strežnika uporabljen ob vsaki zahtevi na strežnik. Mobilna aplikacija je v tem primeru na Sliki 2.9 zahtevala kodo za začetek preverjanja znanja s pomočjo orodja Axios.

```
const getCode = () => async (dispatch) => {
  dispatch(fetchRequest('CODE'))
  try {
    const response = await axios.post(`${urlAdress}/graphql`, { query: codeQuery });
    dispatch(fetchResponse('CODE', response.data.data.exam_code))
  } catch(error) {
    console.error(error);
  }
}
```

Slika 2.9: Zahteva na strežnik z uporabo orodja Axios

2.9.2 Babel

Babel je orodje oziroma prevajalnik, ki omogoča, da je novogeneracijski jezik Javascript ES6 moč zagnati na spletnih brskalnikih ali strežnikih [18]. Prav tako ima vgrajeno podporo za prevajanje Reactove sintakse JSX.

Zelo je priljubljen med razvijalci Reacta saj omogoča uporabo različnih modulov, uporabo zahtevnejših funkcij kot sta *async* in *await*, podporo na različnih brskalnikih in prevajanje sintakse JSX. Kot je vidno na Sliki 2.10, pisanje z uporabo orodja Babel razvijalcu omogoča enostavnejše in bolj pregledno pisanje kode.

```
import './../transitions.css';

const input = [1, 2, 3];
console.log(input.map(item => item + 1));
```

```
"use strict";

require("./../transitions.css");

var input = [1, 2, 3];
console.log(input.map(function (item) {
  return item + 1;
}));
```

Slika 2.10: Primerjava med pisanja kode z orodjem Babel ES2015 (levo) in brez orodja (desno)

2.9.3 Sublime Text 3

Sublime Text je cross-platform urejevalnik izvorne kode [19]. Podpira številne programske jezike in označevalne (ang. markup) jezike. Prav tako je možno uporabljati različne vtičnike, ki vsakemu uporabniku olajšajo delo ali pa mu pri tem pomagajo. Tipično so vtičniki dodani s strani različnih uporabnikov

in so dostopni vsem. Pri pisanju kode smo uporabljali vtičnike za lepše urejanje in formatiranje kode.

Poglavje 3

Implementacija sistema

3.1 Strežnik

3.1.1 Postavitev v oblaku

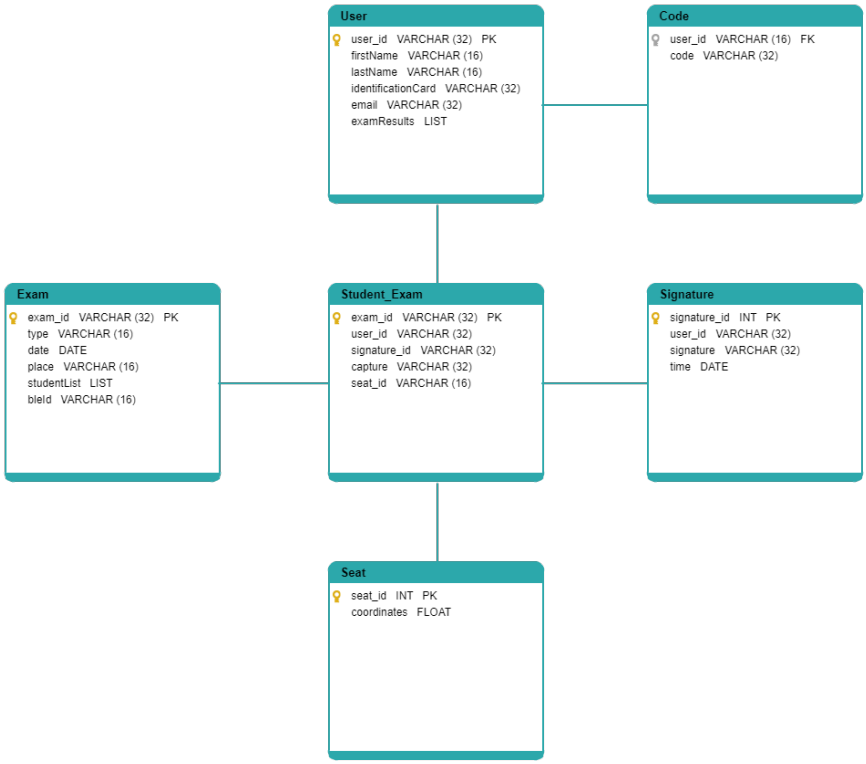
Strežnik je postavljen v oblak s pomočjo sistema Kubernetes. Strežnik je tipa *Deployment*, kot vir kode vzame vnaprej narejeno sliko (ang. image), ki vsebuje vso potrebno kodo in operacijski sistem za delovanje strežnika (glej Dockerfile na Sliki 2.8).

Pogon kode za *Deployment*, prikazane na Sliki 3.1 nam zgolj postavi vsebnik Docker znotraj sistema Kubernetes, ni pa dostopen na svetovnem medmrežju. Za potrebe tega je treba postaviti tako imenovani *Service* (Slika 3.1), ki nam omogoča, da nanj preko vrat (ang. port) 3000 z imenom *main* povežemo svoj strežnik oziroma deployment. Service strežnika dobi svojo številko IP in na tem naslovu je dostopen strežnik.

Slika 3.1: Deployment (levo) in Service (desno)

3.1.2 Shema podatkovne baze

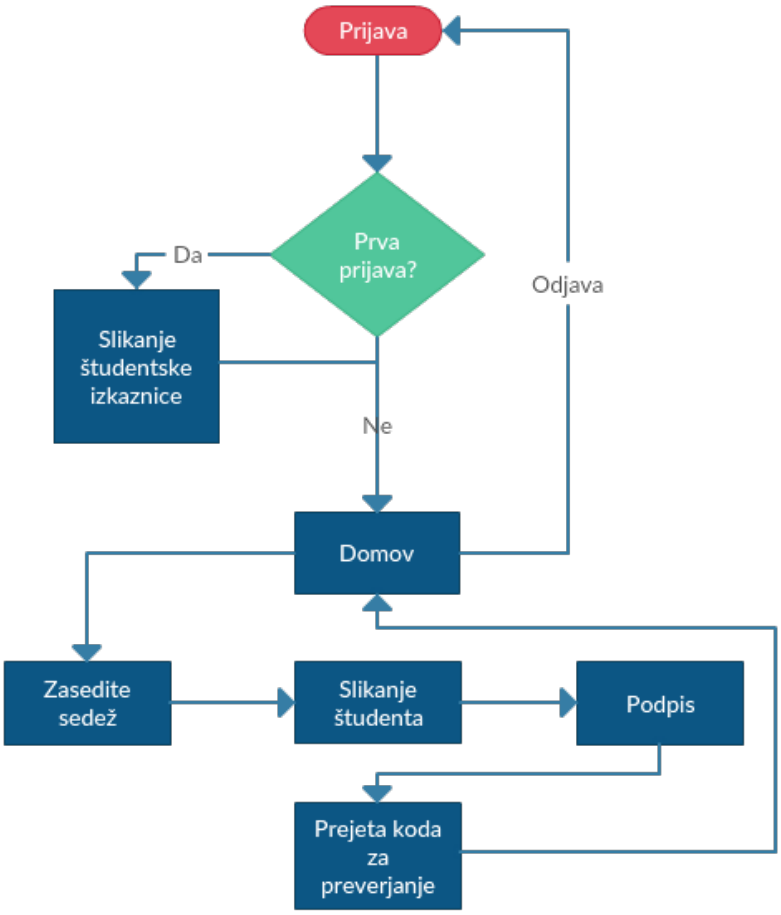
Shema podatkovne baze RethinkDB je sestavljena iz 6 tabel (Slika 3.2). Tabele se povezujejo preko glavne tabele *Student Exam*, ki hrani vse potrebne podatke o študentovi avtentikaciji na določenem izpitu, katerega podatki pa se hranijo v tabeli *Exam*. Tabela podpisov *Signature* je potrebna za hranjenje zgodovine vseh podpisov in kasnejše preverjanje s preteklimi podpisi. Tabela *Seat* zgolj hrani informacijo o koordinatah sedežev znotraj predavalnic, saj je *seat id* že sestavljen iz številke predavalnice in kraja znotraj predavalnice *((številka predevalnice)–vrsta-sedež)*. Tabela *Code* hrani šifro potrebno za začetek preverjanja znanja. Zadnja tabela *User* predstavlja študenta in njegove osebne podatke, skupaj z rezultati prejšnjih preverjanj znanj.



Slika 3.2: Shema podatkovne baze

3.2 Postopek delovanja sistema za uporabnika

Študent si najprej namesti aplikacijo na mobilno napravo. Potek aplikacije je sestavljen iz sedmih komponent, prikazanih na Sliki 3.3. V primeru prve prijave v aplikacijo mora študent slikati študentsko ali osebno izkaznico, šele za tem se mu odpre glavni pogled *Domov*. Pogled predstavlja glavni del aplikacije, ki nudi informacije o prihodnjih in preteklih preverjanjih, kot tudi nadaljevanje na avtentikacijo za aktivno preverjanje znanja. S klikom na začetek avtentikacije se mu odpre pogled s prikazom sedeža, ki ga mora zasesti. Ko ga zasede in ko BLE Beacon zazna študentovo mobilno napravo,



Slika 3.3: Potek delovanja mobilne aplikacije

nadaljuje postopek s slikanjem samega sebe in podpisom na mobilno napravo. Ko opravi postopek avtentikacije, prejme kodo za začetek preverjanja znanja in aplikacija ga preusmeri na domači pogled.

3.2.1 Prijava

Vsak študent si naloži aplikacijo na svojo mobilno napravo. Ob prvem odprtju aplikacije se študentu pojavi prijavno okno, kot je prikazano na Sliki 3.4. Za začetek uporabe aplikacije mora študent vpisati svoj e-naslov, ki ga uporablja na spletni učilnici, in geslo, ki je vpisna številka študenta. Ob

Slika 3.4: Prijavno okno

kliku na gumb PRIJAVA se sproži klic na strežnik, in sicer na URL-naslov *https://api.smolej.ml/auth* (Slika 3.5). Povezava je skrbno zavarovana in zakodirana, saj uporablja kriptografski protokol za izmenjavo podatkov znotraj računalniškega omrežja TLS. Na strežnik se pošljeta tudi vpisan e-naslov in geslo kot spremenljivki *email* in *password* ter poizvedba *loginQuery*, namenjena prijavi, napisana s poizvedovalnim jezikom GraphQL.

```
const loginProcess = (email, password) => async (dispatch) => {
  dispatch(fetchRequest('LOGIN'))
  try {
    const response = await axios.post(`${urlAdress}/auth`, { query: loginQuery, variables: { email, password }});
    dispatch(fetchResponse('LOGIN', response.data.data.login))
  } catch(error) {
    console.error(error);
  }
}
```

Slika 3.5: Koda za prijavo na strani mobilne aplikacije

Ko strežnik prejme zahtevo, se sproži koda na Sliki 3.6. Strežnik preveri, ali lahko najde študenta v tabeli vseh uporabnikov *users* znotraj podatkovne baze RethinkDB. V primeru da študenta ne najde v sistemu ali pa je bilo vpisano geslo napačno, študent prejme opozorilo o napaki pri prijavi.

Ob prvi uspešni prijavi aplikacija prejme žeton (ang. token). Prejme ga

s strani strežnika in je narejen s pomočjo Json Web Tokena. JWT ali JSON Web Token je odprti standard (RFC 7519), ki opredeljuje kompakten in samostojen način za varno prenašanje informacij med strankama kot objekt JSON. Te informacije so digitalno podpisane, zato se jim lahko zaupa. JWT je lahko podpisan s skrivnostjo (ang. secret), z algoritmom HMAC ali z uporabo privatnega in javnega ključa z uporabo RSA. Izbral sem algoritem HS256, ki je narejen na osnovi algoritma HMAC in vsebuje skrivnost izbrano iz naše strani.

Ko aplikacija prejme žeton si ga shrani v svoj pomnilnik. Tako ob vsaki zahtevi na strežnik pošlje tudi svoj prejeti žeton. Žeton omogoča avtentikacijo znotraj aplikacije in dovoljuje dostop zgolj do določenih URL-naslovov. S tem tudi odpravimo ponovno prijavljanje ob vsakem zagonu aplikacije. Študentu se bo tako potrebno prijaviti zgolj enkrat, od tam naprej pa bo žeton poskrbel, da bo uporabnik vedno prijavljen.

```
export const login = async (_, { email, password }) => {
  try {
    // Gets the user by email from the database
    const [user] = await r.table('users')
      .getAll(email, { index: 'email' })
      .coerceTo('array')
      .run(global.rethinkConnection);

    // If user can't be found
    if (!user) return new Error('Email not found!');

    // Checks if password is correct
    if (password !== user.password) return new Error('Password is not correct!');

    // Create a payload for a token
    const payload = {
      id: user.id,
      email: user.email,
    };

    // Create a token and assign the token to user object
    const token = jwt.sign(payload, tokenSecret);

    return { user, token };
  } catch (error) {
    return error;
  }
};
```

Slika 3.6: Koda za prijavo na strani strežnika

3.2.2 Študentska izkaznica

Slika 3.7: Slikanje študentske izkaznice

Po uspeli prijavi se študentu odpre pogled prikazan na Sliki 3.7 in vključi zadnja kamera. Študent mora slikati študentsko ali osebno izkaznico. Slika dokumenta je potrebna zaradi dveh različnih razlogov:

- prijavljeni študent se resnično potrdi in s sliko osebne izkaznice dokaže svojo identiteto,
- v postopku avtorizacije za posamezno preverjanje znanja se bo zajeta slika dokumenta uporabila za primerjanje z zajeto sliko študenta.

Zahteva in slika se pošljeta na naslov *https://api.smolej.ml/graphql*. Ta naslov je zavarovan in uporablja standard JWT. Tako bo zahtevo sprejel samo od odjemalca oziroma študenta, ki je potrjen s strani strežnika in standarda JWT. Za pravilno delovanje moramo JWT postaviti v glavo (ang. header) axios zahteve (Slika 3.8).

Shema poteka prenosa podatkov je v tem primeru identična kot pri slikanju ali podpisu študenta (glej Sliko 3.14).

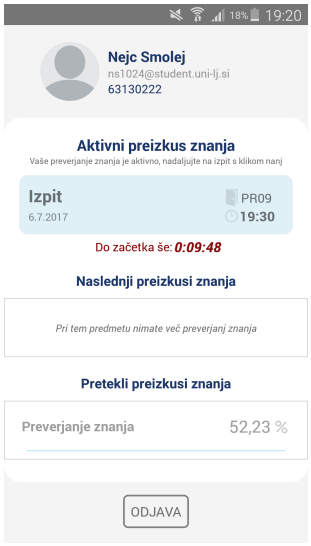
```
render() {
  const { token, user } = this.props

  // If user doesn't have saved token, return Login page
  if (!token) return <LoginView />;

  // Set token as header for axios requests
  axios.defaults.headers.common.Authorization = `JWT ${token}`;

  // If user didnt't take picture of his ID card, return Capture ID page
  if(user.identificationCard === "") return <IdCardView />;

  // Return navigator for application
  return(
    <Navigator
      initialRoute={this.state.routes[0]}
      initialRouteStack={this.state.routes}
      renderScene={this.renderScene}
      configureScene={(route,nav) =>  Navigator.SceneConfigs.PushFromRight}
    />
  )
}
```

Slika 3.8: Koda, ki se izvede ob odprtju aplikacije

3.2.3 Domov

Ob vsakem naslednjem zagonu aplikacije se bo prikazal pogled Domov, prikazan na Sliki 3.9. Tu bo študent dobil vse potrebne informacije o prihajajočih preverjanjih znanja, ocenah in aktivnem preverjanju znanja. Vsa potrebna preverjanja znanja dobi s klicem na strežnik, in sicer na naslov *https://api.smolej.ml/graphql* skupaj z zahtevano poizvedbo *examQuery* (Slika 3.10).

S klikom na *Pretekli preizkusi znanja* se nam odpre celoten seznam vseh opravljenih preverjanj. Poleg tipa preverjanja znanja, ki je lahko preizkus znanja ali pa izpit, bo študent prav tako imel na voljo odstotek doseženih točk na preizkusu.



Slika 3.9: Domov

```
const getExams = async (dispatch) => {
  dispatch(fetchRequest('EXAMS'))
  try {
    const response = await axios.post(`${urlAdress}/graphql`, { query: examsQuery });
    dispatch(fetchResponse('EXAMS', response.data.data.exams))
  } catch(error) {
    console.error(error);
  }
}
```

Slika 3.10: Koda za zahtevo na strani mobilne aplikacije

Pod zavihkom *Naslednji preizkusi znanja* bo na voljo seznam preverjanj znanj, ki sledijo. Posamezni element znotraj seznama je sestavljen iz:

- tipa preverjanja znanja,
- lokacije,
- datuma,
- ure.

Na dan preverjanja znanja, 15 minut pred začetkom, se pod zavihkom *Aktivni preizkus znanja* pojavi preverjanje, ki je aktivno oziroma preverjanje,

ki se bo kmalu začelo za določenega študenta. Informacije o izpitu ostanejo iste kot pri zavihku *Naslednji preizkusi znanja*, le da je dodan odštevalnik do začetka izpita. S klikom na aktivni izpit se prične postopek avtorizacije. Možnost klika je omejena na 15 minut pred začetkom preverjanja in 30 minut po začetku.

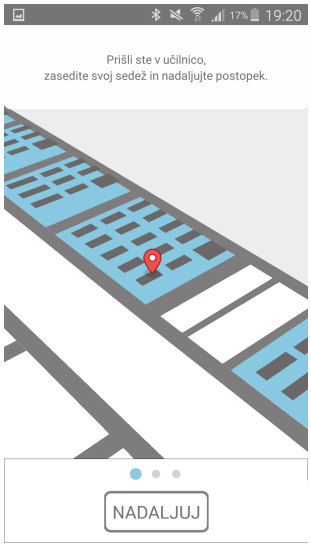
3.2.4 Avtorizacija na preverjanju znanja

Vsak profesor ali asistent, ki nadzoruje določeno preverjanje znanja prejme BLE Beacon. Vsak BLE Beacon je vnaprej konfiguriran za potrebe delovanja znotraj predavalnice. Radij oddajanja signala Bluetooth je nastavljen na 10m. Tako profesor ob vstopu v predavalnico napravo postavi na svojo mizo in ta bo oddajala signale.

Avtorizacija na preverjanju znanja je sestavljena iz treh korakov:

1. zemljevida,
2. slikanje obraza,
3. podpisa študenta.

1. Zemljevid predavalnice



Slika 3.11: Sedežni red

Študentu se prikaže zemljevid predavalnice, v kateri se izvaja preverjanje znanja (Slika 3.11). Prav tako je z rdečim simbolom (ang. pin) označen sedež, na katerega se mora študent usesti. Podoba zemljevida je narejena s

pomočjo Mapboxa, platforme, ki omogoča samostojno kreiranje poligonov, črt in pik na zemljevidu [20].

Prva naloga študenta v postopku avtorizacije je zasedanje mesta znotraj predavalnice. Vsak študent mora najprej prižgati Bluetooth. Ko je Bluetooth na mobilni napravi priključen, aplikacija začne iskati signale Beacona (Slika 3.12). Šele ko aplikacija zazna Beacon, to pomeni, da sta študent in njegova mobilna naprava znotraj predavalnice. Takrat lahko študent zasede svoje mesto in nadaljuje postopek.

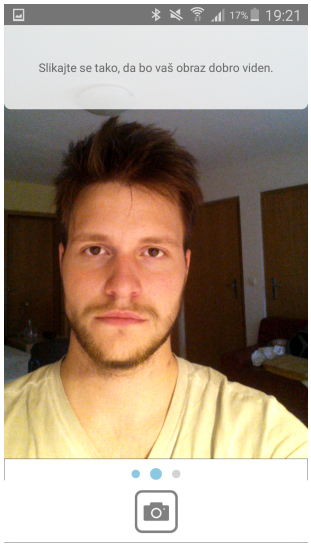
```
handleDiscoverPeripheral(peripheral) {
  if(peripheral.name === this.props.currentExam.bleId) {
    // Student is in classroom
    if(!this.state.isNear) {
      this.setState({
        isNear: true,
      })

      // Slide down notification saying the BLE Beacon has been found
      Animated.timing(this.state.notificationY, {toValue: 0}).start();
      this.props.canContinue();

      // Stop scaning for BLE Beacon
      clearInterval(scanInterval);
    }
  }
}
```

Slika 3.12: Koda za zaznavanje BLE Beacona na strani mobilne aplikacije

2. Slikanje obraza

V drugem koraku se študentu odpre sprednja kamera in gumb za slikanje. Študent bo s klikom na gumb zajel sliko in jo posredoval na strežnik.

Strežnik zajeto sliko osebne izkaznice posreduje hranilniku, imenovanem Google Cloud Storage. Pot shranjene slike znotraj hranilnika pa nato shrani v podatkovno bazo RethinkDB (Slika 3.14). Slika bo v trenutku vidna na pregledni plošči profesorja oziroma nadzornika.

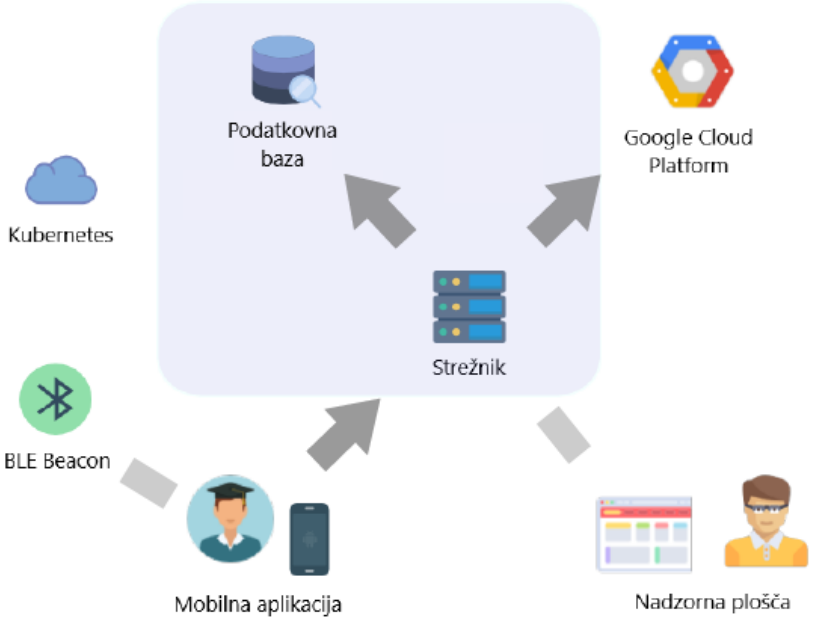


Slika 3.13: Slikanje obraza

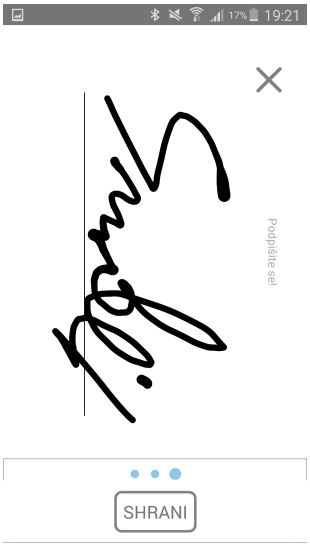
3. Podpis

Tretji in zadnji korak je podpis študenta. Študent se mora podpisati na zaslon mobilne naprave, kot je prikazano na Sliki 3.15. Ob kliku na gumb SHRANI se slika podpisa pošlje na strežnik in profesorju na pregledno ploščo. V primeru, da se študent zmoti in se želi ponovno podpisati lahko pritisne gumb s simbolom x, ki pobriše dosedanji podpis.

Po uspešni celotni avtentikaciji študent prejme kodo, ki mu služi kot vstopna koda za preverjanje znanja ali izpit (Slika 3.16).



Slika 3.14: Shema poteka prenašanja podatkov pri slikanju obraza študenta

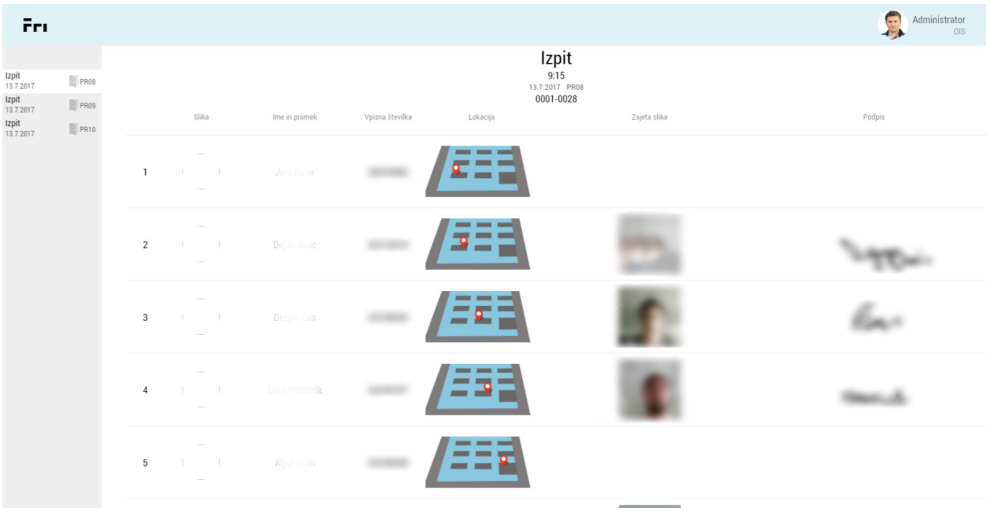


Slika 3.15: Podpis

Slika 3.16: Koda za preverjanje znanja

3.3 Postopek delovanja sistema za administratorja

3.3.1 Pregledna stran



Slika 3.17: Pregledna stran

Ob odprtju nadzorne plošče ob strani vidimo stransko vrstico, v kateri lahko izbiramo izpit. V primeru, da se izpit pri predmetu izvaja v več predavalnicah, je stranska vrstica še toliko bolj priročna. Z enim klikom na preverjanje znanja lahko vidimo stanje avtentikacije študentov v sosednjih predavalnicah, prikazano na Sliki 3.17.

Ko profesor izbere preverjanje znanja, se mu izpišejo podatki o preverjanju in seznam prijavljenih študentov. Poleg osnovnih podatkov o preverjanju znanja, kot so tip preverjanja, kraja in časa, je moč videti identifikacijsko številko BLE Beacona, potrebnega za to predavalnico. Ključno je, da se ta ujema z identifikacijsko številko BLE Beacona, ki ga je prejel nadzornik preverjanja znanja.

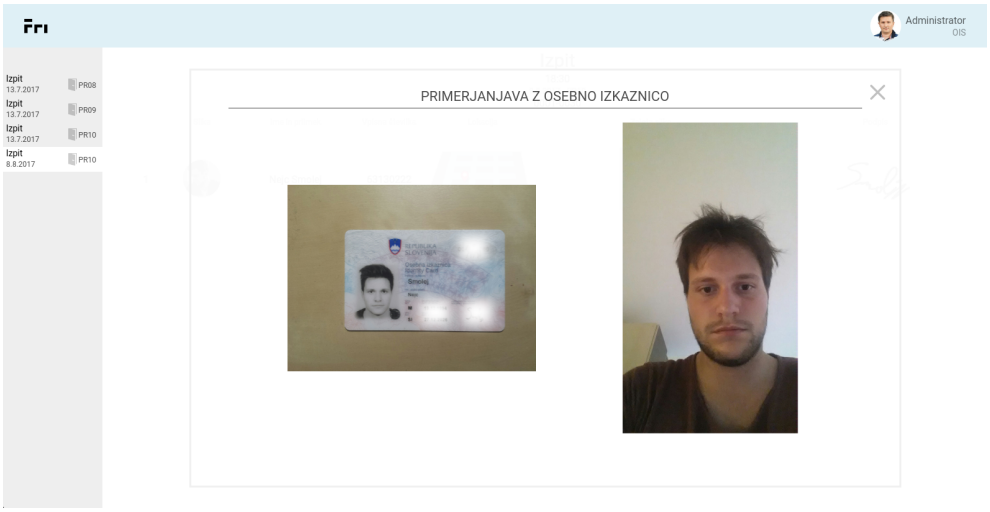
Vsak element v seznamu študentov je sestavljen iz naslednjih komponent:

- profilna slika,

- ime in priimek,
- vpisna številka,
- lokacija študenta,
- zajeta slika,
- zajet podpis.

Seznam študentov se osvežuje na vsake 4 sekunde. V primeru, da se je študent pravkar slikal ali podpisal, se zajeta slika oziroma podpis pojavita na seznamu. V primeru, da se študent ne avtenticira je lahko razvidno, na katerem sedežu sedi ta oseba in je na to lahko hitro opozorjena.

3.3.2 Primerjava zajete slike



Slika 3.18: Pregledna stran

Profesor lahko vsakega študenta preveri tudi sam. S klikom na zajeto sliko študenta se mu odpre okno *Primerjava zajete slike* (Slika 3.18). Okno vsebuje zajeto sliko na preverjanju znanja in sliko študentske oziroma osebne izkaznice študenta. Profesor tako lahko še sam preveri ujemanje slike na osebni izkaznici ter zajete slike študenta.

3.3.3 Primerjava podpisa

Slika 3.19: Pregledna stran

Ob kliku na podpis študenta se odpre novo okno *Primerjava podpisa* (Slika 3.19). Okno vsebuje trenutni podpis in zadnjih 5 podpisov izbranega študenta. Tudi tako lahko profesor glede na zgodovino podpisov vidi, ali se podpis študenta ujema s preteklimi.

Poglavje 4

Testiranje

4.1 Priprava na testiranje

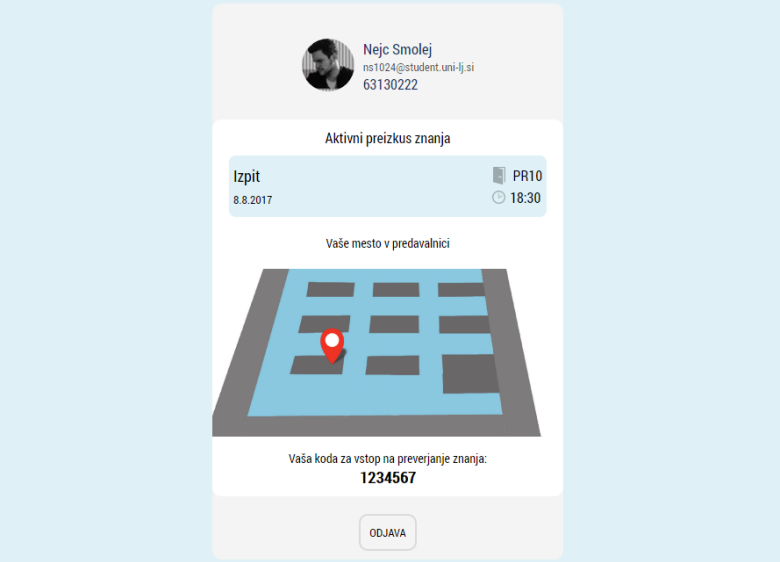
4.1.1 Spletna verzije mobilne aplikacije

Testiranje mobilne aplikacije je bilo možno zgolj za uporabnike mobilne platforme Android. V ta namen smo za študente s platformami iOS in Windows Phone naredili spletno stran *https://www.smolej.ml* (Slika 4.1), ki nudi spletno različico okrnjene mobilne aplikacije.

Po uspešni prijavi študenta z vpisom e-naslova in gesla spletna stran nudi zgolj podatke, potrebne za uspešen vstop na izpit:

- osnovne podatki o začetku preverjanja,
- mesto v predavalnici,
- kodo za dostop do preverjanja.

Prav tako je bil za lažjo navigacijo po aplikaciji narejen vodič za študente, ki ga je bilo moč najti na spletnem naslovu *https://www.smolej.ml/pomoc*. Vodič je vseboval vseh 7 korakov, od prijave do prejetja kode potrebne za preverjanje znanja.



Slika 4.1: Glavna stran spletne aplikacije

4.1.2 Konfiguracija BLE Beacona

Vsi nadzorniki so pred začetkom preverjanja znanja dobili BLE Beacone. Vsak izmed BLE Beaconov je bil konfiguriran za potrebe delovanja znotraj dodeljene predavalnice. Osnovni podatki o uporabljeni strojni opremi, BLE Beaconu (Slika 2.3):

- proizvajalec: Accent Systems,
- model: iBKS Plus,
- maksimalna možna življenjska doba baterije: 105 mesecev,
- maksimalna možnost doseg: 70 m,
- čas obveščanja: 100 ms,
- radijski prenos moči: -8.

4.1.3 Sinhronizacija podatkov profesorja s podatkovno bazo

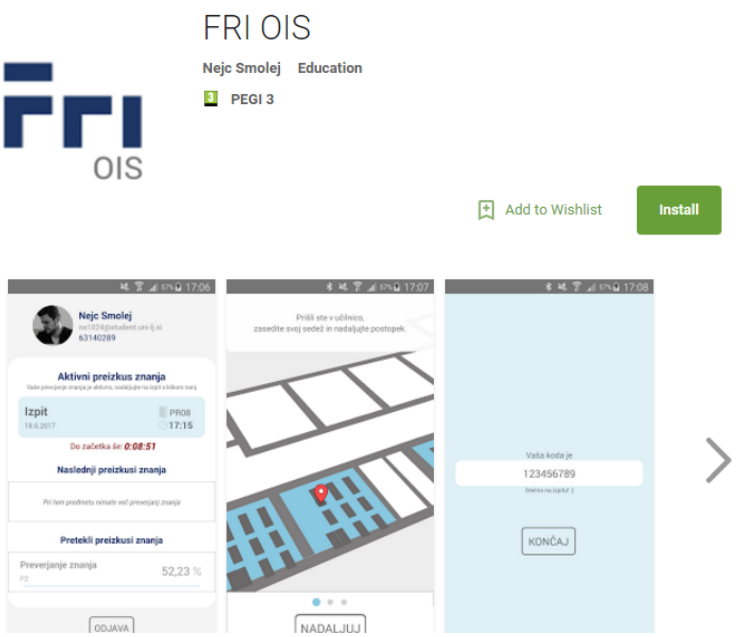
Vsi pomembni podatki o študentih so bili posredovani preko Google Preglednic [21]. Skupaj z mentorjem si delimo skupno preglednico, do katere imamo dostop. Za potrebe sinhroniziranja podatkovne baze s podatki, posredovanimi s strani profesorja, je bila napisana koda (Slika 4.2), ki je vse študente pretvorila v objekte JSON, primerne za uvoz. Medtem pa so bili podatki o preverjanju znanja, kot so kraj, čas, identifikacijska številka BLE Beacona in sedežni red, vneseni kar ročno.

```
const { OAuth2 } = google.auth;

const oauth2Client = new OAuth2(
  new_client_id,
  new_client_secret,
  new_redirect_uris
);

const main = async () => {
  oauth2Client.setCredentials(new_token);

  GoogleSpreadsheets.rows({
    key: '1PAgyJvkWkRJfiTRWGIpbRlicTF97fJ2MzuEdmzXzD1Q',
    auth: oauth2Client,
    worksheet: 'od6',
  }, (err, spreadsheet) => {
    console.log(spreadsheet[0])
    const students = spreadsheet.map(row => ({
      id: row['vš'],
      firstName: row['ime'].charAt(0) + row['ime'].slice(1).toLowerCase(),
      lastName: row['priimek'].charAt(0) + row['priimek'].slice(1).toLowerCase(),
      email: row['email.eucilnica'],
      exam_final_result: row['pzskupaj'],
      exam_results: [
        {
          result: row['pz1skupaj'],
          lost_focus: row['pz1odbitek'],
        },
        {
          result: row['pz2skupaj'],
          lost_focus: row['pz2odbitek'],
        },
        {
          result: row['pz3skupaj'],
          lost_focus: row['pz3odbitek'],
        },
        {
          result: row['pz4skupaj'],
          lost_focus: row['pz4odbitek'],
        },
      ],
      identificationCard: '',
    }));
    fs.writeFile('./src/parsers/students.json', JSON.stringify(students, null, 2), (err) => {
      if(err) console.error(err);
    })
  });
}

main();
```

Slika 4.2: Koda, potrebna za sinhronizacijo podatkov

4.1.4 Google Play

Mobilna aplikacija je objavljena v trgovini mobilnih aplikacij za mobilno platformo Android, Google Play (Slika 4.3). Študentje aplikacijo najdejo pod imenom FRI OIS.



Slika 4.3: Mobilna aplikacija, objavljena v mobilni spletni trgovini Google Play

4.2 Dan testiranja - preverjanje znanja

Testiranje je potekalo 12. 7. 2017 v času preverjanja znanja pri predmetu Osnove informacijskih sistemov. To je bilo drugo opravljanje končnega izpita pri tem predmetu, na izpit je bilo prijavljenih 52 študentov. Zato je preverjanje potekalo v treh predavalnicah, in sicer v P8, P9 in P10.

Na dan preverjanja znanja si je izmed 52 prijavljenih študentov, aplikacijo naložilo 24 študentov, ki imajo mobilno platformo Android. 15 minut pred začetkom preverjanja znanja so prvi študentje vstopili v predavalnico.

Študentje so bili zmedeni glede delovanja aplikacije, saj se nihče ni uspel prijaviti v aplikacijo, kljub temu da so imeli objavljena navodila za uporabo v spletni učilnici in v spletnem vodiču, objavljenem na naši spletni strani. Ob ponovnem opozorilu profesorja, da je geslo enako vpisni številki posameznika in ne geslu v spletni učilnici, so se prvi študentje že lahko prijavili v mobilno aplikacijo. Ta nesporazum je povzročil manjši upad uporabnosti aplikacije, saj so študentje imeli na voljo manj kot 15 minut za postavitev okolja, uspešno prvo prijavo s slikanjem študentske izkaznice in avtentikacijo na preverjanju znanja.

Kljub manjšemu nesporazumu so študentje začeli postopek avtentikacije in prvi rezultati so že bili vidni na nadzorni plošči. Skupaj z mentorjem smo na nadzorni plošči spremljali prve zajete slike študentov in njihove podpise. Do začetka preverjanja se je uspešno avtenticiralo kar 15 študentov, kar je dober podatek glede na dejstvo, da še 15 minut pred začetkom preverjanja niti eden izmed 24 študentov ni bil prijavljen v mobilno aplikacijo.

Ocenjujemo, da so študentje za avtentikacijo potrebovali v povprečju 45 sekund. Vse to vključuje vklop Bluetootha ob vstopu v predavalnico, slikanje študenta in podpis študenta. Za primerjavo smo vzeli nadzornika, ki bi za avtentikacijo enega študenta potreboval 12 sekund, kar vključuje prihod do študenta, iskanje študenta na seznamu vseh prijavljenih, pregled študentske izkaznice in študenta. Če je na preverjanje prijavljenih 50 študentov, bi za celoten pregled študentov potreboval kar 10 minut.

4.3 SWOT analiza

Slika 4.4: SWOT analiza

Poglavje 5

Zaključek

5.1 Sklepne ugotovitve

Izboljšan sistem za preverjanje prisotnosti na preverjanju znanja in prototip mobilne aplikacije sta doživela prvo testiranje. Več kot polovica študentov (natančneje 15 od 24) z nameščeno aplikacijo je končalo postopek avtentikacije in prejelo kodo za preverjanje znanja. Pri tem podatku ne smemo zanemariti dejstva, da je vsak izmed 24-ih uporabnikov aplikacije avtentikacijo lahko po svoji želji dokončal tudi na spletni strani, ki je zgolj podala sedežni red študenta in potrebno kodo za preverjanje znanja.

Prav tako kot del odzivnosti študentov je bil podatek, da so študentje za avtentikacijo potrebovali manj kot 1 minuto za nas dokaz, da je aplikacijo možno uporabljati pred preverjanjem in da ni preobremenjevala študentov. Če primerjamo s časom, ki ga zapravi nadzornik za preverjanje prisotnosti, je razlika več kot očitna. Čas, ki ga zapravi študent za potrditev samega sebe na preverjanju znanja, je ničen v primerjavi s časom, ki ga porabi nadzornik, da preveri vse študente. Ne smemo pozabiti da nadzornik med hojo po predavalnici in preverjanjem študentskih izkaznic ne mora nadzorovati študentom med pisanjem. Med tem časom lahko nekateri že izkoristijo nezbranost nadzornika in goljufajo na preverjanju znanja.

Nekateri nadzorniki preverjajo študente zelo površno, zgolj s klicanjem

njihovih imen. Z uporabo aplikacije bi bil ta problem odpravljen, vsak študent bi bil potrjen, nadzornik pa ne bi moral zapravljati nepotrebnega časa s hojo po velikih predavalnicah.

Vseeno ne smemo pozabiti na dejstvo, da se z uporabo te aplikacije obveznosti prenesejo na študenta, medtem ko se delo nadzornika razbremeni. Prav tako lahko študentova nervoza pred začetkom preverjanja znanja vpliva na slabo izkušnjo in slabo uporabnost aplikacije. Vendar, kadar gre za preverjanje, ali je študent ta, za katerega se izdaja in ali je možnost, da odkrijemo prepisovalca, smo mnenja, da je tu potrebna maksimalna možna zaščita.

5.2 Možne nadgradnje

Mobilna aplikacija za platformo iOS

Sprva bi mobilno aplikacijo bilo treba dokončati tudi za platformo iOS. Nekaj osnovnih funkcij aplikacije sicer že deluje in ta je že bila testirana na eni mobilni napravi, vendar zaradi potrebe po razvijalskem okolju z operacijskim sistemom Mac to ni bilo povsem mogoče. Z razvito aplikacijo za platformi Android in iOS bi pokrili velik del vseh uporabnikov mobilnih naprav. S tem bi odpravili spletno verzijo mobilne aplikacije in prav vsi študentje bi bili primorani uporabljati mobilno aplikacijo.

STUDIS API

Vsi podatki, podani s strani profesorja, so bili izmenjani preko Google Preglednic. Za pregled prijavljenih študentov na preverjanje znanja bi bilo veliko bolje dostopati do samega vira teh podatkov, Študijskega informacijskega sistema, imenovanega STUDIS. To je možno narediti z dostopom do STUDIS API-ja, za katerega pa bi potrebovali posebno dovoljenje za zgolj branje vsebine podatkov.

Google prijava

Prijavo v aplikacijo bi močno olajšali z implementacijo Google prijave v sistem. Vsak študent lahko dostopa do svojega študijskega e-računa kar z uporabo Googla. Z uporabo Google prijave bi študentje zgolj s klikom vstopili v aplikacijo, ne da bi bili primorani vpisovati e-naslov in geslo. Proces bi lahko še bolj nadgradili z uporabo Active Directory-ja [22]. Storitev Active Directory nam omogoča grupiranje prijavljenih oseb v skupine z različnimi dovoljenji. Tako bo povsem enostavno videti, kdo je nadzornik (za uporabo nadzorne plošče) in kdo študent (za uporabo mobilne aplikacije).

Gumb Potrebujem pomoč

Nemalokrat se zgodi, da morajo študentje zaradi vprašanja ali nejasnosti na preverjanju dlje časa držati roko, da pritegnejo pozornost profesorja. V času preverjanja znanja bi lahko študent imel gumb POTREBUJEM POMOČ, ki bi oznanjal, da študent potrebuje pozornost profesorja za pomoč. Profesor bi v trenutku zahteve pomoči prejel obvestilo na nadzorno ploščo. Prav tako bi bilo povsem vidno, kateri študent zahteva pomoč in na katerem mestu in v kateri predavalnici sedi. V primeru, da preverjanje poteka v več predavalnicah, nadzornik, ki je zmožen odgovoriti na vprašanje študenta, pa je zgolj eden, bi to bil zelo dober način kako v najkrajšem času pridobiti pozornost nadzornika in posledično pridobiti odgovor na željeno vprašanje.

Obveščanje o prejeti oceni

V aplikacijo bi lahko vgradili možnost obveščanja o prejeti novi oceni. Tako bi vsa komunikacija glede prejetih ocen potekala zgolj preko mobilne aplikacije, študentje pa bi bili o prejeti oceni obveščeni v trenutku preko tako imenovanih push obvestil.

5.3 Prihodnost sistema - FRI Asistent

Sistem je bil razvit samo v okviru predmeta Osnove informacijskih sistemov. V prihodnosti bi lahko sistem razvili za celotno fakulteto. Naša vizija je, da bi vsak študent imel naloženo aplikacijo, ki bi se imenovala FRI Asistent. Aplikacija bi bila stik študenta s celotnim sistemom vseh izpitov. Imeli bi vse podatke o prihajajočih izpitih in preverjanjih znanja. Lahko bi pregledovali ocene in bili obveščeni o objavi novih. Potrjevali bi se na vseh izpitih, medtem ko bi jih profesorji in asistenti nadzirali preko nadzorne plošče.

Končna vizija mobilne aplikacije in sistema je, da bi aplikacija postala standard preverjanja prisotnosti na Fakulteti za računalništvo in informatiko. Da bi v predavalnici na preverjanju znanja vsak prijavljeni študent zasedel mesto znotraj predavalnice in potrdil prisotnost, ne da bi s tem obremenjeval nadzornika. Medtem ko nadzornik ne bi zapravljal časa s presedanjem študentov in klicanjem njihovih imen, bi študentje svojo prisotnost potrdili ob vstopu v predavalnico z uporabo mobilne aplikacije, ne da bi čakali na ostale študente in nadzornika, da jih preveri. S tem bi se na vseh preverjanjih močno pospešil in izboljšal postopek preverjanja identifikacije študentov.

Literatura

[1] "Tržni delež mobilnih operaterjev." `https://www.mobilna-telefonija.com/informator/trzni-delez/mobilni-operaterji.html`. Accessed: 2017-08-10.

[2] D. Koštomaj, *Izkaznica NFC v postopku preverjanja študijskih obveznosti*. PhD thesis, Univerza v Ljubljani, 2014.

[3] D. Lebar, *Prisotnost pri pouku s pomočjo RFID*. PhD thesis, Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, 2014.

[4] L. IndoorAtlas, "Ambient magnetic field-based indoor location technology: Bringing the compass to the next level," *IndoorAtlas Ltd*, 2012.

[5] B. Eisenman, *Learning React Native: Building Native Mobile Apps with JavaScript*. "O'Reilly Media, Inc.", 2015.

[6] "Axios, promise based http client for the browser and node.js." `https://github.com/mzabriskie/axios`. Accessed: 2017-08-11.

[7] A. Fedosejev, *React. js Essentials*. Packt Publishing Ltd, 2015.

[8] "Electron." `https://electron.atom.io`. Accessed: 2017-08-11.

[9] N. Newman, "Apple ibeacon technology briefing," *Journal of Direct, Data and Digital Marketing Practice*, vol. 15, no. 3, pp. 222–225, 2014.

[10] G. Tiepolo, *Getting Started with RethinkDB*. Packt Publishing Ltd, 2016.

[11] A. Mardan, *Express. js Guide: The Comprehensive Book on Express. js*. Azat Mardan, 2014.

[12] "Express.js." `https://expressjs.com/`. Accessed: 2017-08-23.

[13] "Who's using graphql." `http://graphql.org/users/`. Accessed: 2017-08-11.

[14] "Graphql." `https://en.wikipedia.org/wiki/GraphQL`. Accessed: 2017-08-11.

[15] C. Sanchez, "Scaling docker with kubernetes," *Website. Available online at http://www. infoq. com/articles/scaling-docker-with-kubernetes*, p. 35, 2015.

[16] "Google cloud storage vs aws s3." `https://www.linkedin.com/pulse/google-cloud-storage-vs-aws-s3-jishnu-kinwar`. Accessed: 2017-08-15.

[17] "Compute engine." `https://cloud.google.com/compute/`. Accessed: 2017-08-11.

[18] "Babel." `https://babeljs.io/`. Accessed: 2017-08-11.

[19] "Sublime text." `https://www.sublimetext.com/`. Accessed: 2017-08-11.

[20] C. Cadenas, "Geovisualization: Integration and visualization of multiple datasets using mapbox," 2014.

[21] "Google sheets." `https://www.google.com/sheets/about/`. Accessed: 2017-08-22.

[22] "Active directory." `https://en.wikipedia.org/wiki/Active_Directory`. Accessed: 2017-08-31.